REMARKS/ARGUMENTS

Reconsideration of this application is respectfully requested.

The rejection of claims 1, 4, 5 and 6 under 35 U.S.C. §102 as allegedly anticipated by Yohe '943 is respectfully traversed.

In a nutshell, applicant's claims deny client access to a server file if the digital signature associated with that file is invalid while Yohe '943 permits client access to a server file if a hash function comparison (apparently what the Examiner is considering to be a digital signature) fails. Yohe '943 is merely directed towards increasing the speed of cache data supply systems while maintaining the most current version of a master file data in each cache location. Hash functions are used to provide a quick technique to check for identical file content. Cryptographic digital signatures are not even utilized.

To the extent that Yohe has any relevance whatsoever, it actually teaches something that would be contrary to the applicant's claimed arrangements where a requested file is not provided if digital signature comparison processes fail.

Yohe has a different aim from applicant's invention. The object of Yohe is to increase the speed with which a remote client computer 12 can access data and directories normally stored on a file server computer 18 (see column 2, lines 24 and 25). Yohe is especially useful if the connection speed from the remote client computer 12 and a file server computer 18 is low (see column 2, lines 9 to 18).

- 9 -

952877

Yohe increases the speed with which the user of the remote client computer 12 is able to access a file by the well-known device of caching (i.e., storing) a copy of recently accessed files in the remote client's memory (30 to 34). When the user of the remote client computer 12 later asks for the same file then it can be provided from the remote client computer's memory (30 to 34) rather than having to be downloaded from the file server computer 18 over the slow modem link (column 2, lines 9 to 16 again).

A problem with caching is that there is no guarantee that the file stored in the user's computer's memory is up-to-date. Yohe checks the file is up-to-date by having the cache verifying computer 14 calculate what Yohe calls a "signature" of the file. In fact, as will be explained below, what is calculated I not a digital signature as that term is normally used in the art, but a "hash value" or "message digest". A hash value or message digest is typically much shorter than the file from which the hash value is generated – in addition, the value is very sensitive to any change in the file from which it is generated. These points are explained in the attached pages copied from the book "Applied Cryptography" referenced at lines 4-6 on page of the applicant's specification.

Thus, if the user of the remote client computer 12 requests a file that is in the remote client computer's memory (30 or 35). Yohe has the remote client 12 calculate a first hash value from the file in the cache (see column 6, lines 22 and 23). The calculated value is sent to the Cache Verifying Computer 14 as part of a VERIFY request (column 6, lines 23 to 26). The Cache Verifying Computer 14 then calculates a second hash value from the version of the file stored at the file server computer 18 (column 6, lines 27 to 29).

- 10 -

The Examiner will realize that if the file cached at the remote client computer 12 is up-to-date then it will be the same as the file provided by the file server computer 18 and used by the Cache Verifying Computer 14 to calculate the second hash value. Since the two files are the same, then it follows that the first and second hash values will be the same.

Common sense tells us that if the two files are the same, then there is no need to update the cache on the remote client computer 12. All that is needed is for the Cache Verifying Computer 14 to send a message that the copy of the file at the remote client computer 12 is up-to-date. Unsurprisingly, this is what Yohe does, column 6, lines 35 to 37. In that case, the cache provides the usual performance enhancement in that the user of the remote client computer 12 des not need to wait for the file to be transmitted over the slow modem link 22, instead the only delay is for the transmission of the shorter message indicating that the cached copy of the file is up-to-date.

The above explanation shows how Yohe achieves the benefits of caching without risking presenting the user of the remote client computer 12 with an out-of-date copy of the requested file.

A consideration of the operation of Yohe and the applicant's claimed arrangements reveals a crucial distinction. In Yohe, if the file has been altered, then the file is transmitted by applicant's server computer 18. That is the complete opposite of the invention claimed in claims 1, 4, 5 and 6 where, if the file has been altered, then it is not transmitted by the sever computer.

Also, as mentioned above, Yohe does not use "digital signatures" as that term is normally understood by those in the art. The fact that what Yohe provides are in fact hash values can be

- 11 -

seen from the references to specific algorithms later on in the Yohe patent (specifically at column 11, line 63 and column 13, line 39). The two RFC's referred to are available via http://www.ietf.org/rfc.html - note that the second reference should be to RFC 1321 not RFC 1322.

The rejection of claim 2 under 35 U.S.C. §103 as allegedly being made "obvious" based on Yohe '943 in view of Atkinson '012 is also respectfully traversed.

Fundamental distinctions over the primary Yohe reference with respect to parent claim 1 have already been noted above.

Atkinson '012 seems to correspond closely to the Authenticode ® software described in the last paragraph of page 5 of the applicant's specification. As pointed out in the first paragraph on page 6 of the specification, the mere fact that the executable file is provided to the client computer is enough to present a security problem which the present invention overcomes by denying the client computer access to the file on the server computer.

Neither Yohe nor Atkinson suggest that an invalid signature should prevent the transmission of data from a server computer to a client computer. As explained above, if anything, Yohe teaches the opposite. Since neither reference teaches the last feature of claim 1, the combination of the two would not render claim 1 obvious, let alone claim 2.

The rejection of claim 3 under 35 U.S.C. §103 as allegedly being made "obvious" based on Yohe '943 in view of Farber '791 is also respectfully traversed.

952877

Once again, fundamental deficiencies of the primary Yohe reference have already been noted above with respect to parent claim 1.

Farber '791 teaches that a file may "expire" at a date and time stored in relation to that file – the file will then be deleted. The Examiner claims that modifying Yohe by introducing this idea from Farber would lead to the invention claimed in claim 3. It would not – the Examiner has not shown that Farber or Yohe teaches the last feature of claim 1.

Accordingly, this entire application is now believed to be in allowable condition and a formal Notice to that effect is respectfully solicited.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: _____
Larry S. Nixon
Reg. No. 25,640

LSN:vc
1100 North Glebe Road, 8th Floor
Arlington, VA 22201-4714
Telephone: (703) 816-4000
Facsimile: (703) 816-4100

952877

he is with the one-way function.) For public-key cryptography, we need something else (although there are cryptographic applications for one-way functions—see Section 3.2).

A **trapdoor one-way function** is a special type of one-way function, one with a secret trapdoor. It is easy to compute in one direction and hard to compute in the other direction. But, if you know the secret, you can easily compute the function in the other direction. That is, it is easy to compute $f(x)$ given x, and hard to compute x given $f(x)$. However, there is some secret information, y, such that given $f(x)$ and y it is easy to compute x.

Taking a watch apart is a good example of a trap-door one-way function. It is easy to disassemble a watch into hundreds of minuscule pieces. It is very difficult to put those tiny pieces back together into a working watch. However, with the secret information—the assembly instructions of the watch—it is much easier to put the watch back together.

2.4   ONE-WAY HASH FUNCTIONS

A **one-way hash function** has many names: compression function, contraction function, message digest, fingerprint, cryptographic checksum, message integrity check (MIC), and manipulation detection code (MDC). Whatever you call it, it is central to modern cryptography. One-way hash functions are another building block for many protocols.

Hash functions have been used in computer science for a long time. A hash function is a function, mathematical or otherwise, that takes a variable-length input string (called a **pre-image**) and converts it to a fixed-length (generally smaller) output string (called a **hash value**). A simple hash function would be a function that takes pre-image and returns a byte consisting of the XOR of all the input bytes.

The point here is to fingerprint the pre-image: to produce a value that indicates whether a candidate pre-image is likely to be the same as the real pre-image. Because hash functions are typically many-to-one, we cannot use them to determine with certainty that the two strings are equal, but we can use them to get a reasonable assurance of accuracy.

A one-way hash function is a hash function that works in one direction: It is easy to compute a hash value from pre-image, but it is hard to generate a pre-image that hashes to a particular value. The hash function previously mentioned is not one-way: Given a particular byte value, it is trivial to generate a string of bytes whose XOR is that value. You can't do that with a one-way hash function. A good one-way hash function is also **collision-free**: It is hard to generate two pre-images with the same hash value.

The hash function is public; there's no secrecy to the process. The security of a one-way hash function is its one-wayness. The output is not dependent on the input in any discernible way. A single bit change in the pre-image changes, on the average, half of the bits in the hash value. Given a hash value, it is computationally unfeasible to find a pre-image that hashes to that value.

from Applied Cryptography - full reference at p4 lines 4 to 6.

Think of it as a way of fingerprinting files. If you want to verify that someone has a particular file (that you also have), but you don't want him to send it to you, then ask him for the hash value. If he sends you the correct hash value, then it is almost certain that he has that file. This is particularly useful in financial transactions, where you don't want a withdrawal of $100 to turn into a withdrawal of $1000 somewhere in the network. Normally, you would use a one-way hash function without a key, so that anyone can verify the hash. If you want only the recipient to be able to verify the hash, then read the next section.

Message Authentication Codes

A message authentication code (MAC), also known as a data authentication code (DAC), is a one-way hash function with the addition of a secret key (see Section 18.14). The hash value is a function of both the pre-image and the key. The theory is exactly the same as hash functions, except only someone with the key can verify the hash value. You can create a MAC out of a hash function or a block encryption algorithm; there are also dedicated MACs.

2.5   COMMUNICATIONS USING PUBLIC-KEY CRYPTOGRAPHY

Think of a symmetric algorithm as a safe. The key is the combination. Someone with the combination can open the safe, put a document inside, and close it again. Someone else with the combination can open the safe and take the document out. Anyone without the combination is forced to learn safecracking.

In 1976, Whitfield Diffie and Martin Hellman changed that paradigm of cryptography forever [496]. (The NSA has claimed knowledge of the concept as early as 1966, but has offered no proof.) They described **public-key cryptography**. They used two different keys—one public and the other private. It is computationally hard to deduce the private key from the public key. Anyone with the public key can encrypt a message but not decrypt it. Only the person with the private key can decrypt the message. It is as if someone turned the cryptographic safe into a mailbox. Putting mail in the mailbox is analogous to encrypting with the public key; anyone can do it. Just open the slot and drop it in. Getting mail out of a mailbox is analogous to decrypting with the private key. Generally it's hard; you need welding torches. However, if you have the secret (the physical key to the mailbox), it's easy to get mail out of a mailbox.

Mathematically, the process is based on the trap-door one-way functions previously discussed. Encryption is the easy direction. Instructions for encryption are the public key; anyone can encrypt a message. Decryption is the hard direction. It's made hard enough that people with Cray computers and thousands (even millions) of years couldn't decrypt the message without the secret. The secret, or trapdoor, is the private key. With that secret, decryption is as easy as encryption.

This is how Alice can send a message to Bob using public-key cryptography:

(1) Alice and Bob agree on a public-key cryptosystem.